A new secant method for unconstrained optimization (draft)

Stephen A. Vavasis*

August 18, 2008

Abstract

We present a gradient-based algorithm for unconstrained minimization derived from iterated linear change of basis. The new method is equivalent to linear conjugate gradient in the case of a quadratic objective function. In the case of exact line search it is a secant method. In practice, it performs comparably to BFGS and DFP and is sometimes more robust.

1 Iterated linear change of basis

We consider the problem of minimizing a differentiable function $f : \mathbf{R}^n \to \mathbf{R}$ with no constraints on the variables. We propose the following algorithm for this problem. We assume a starting point $\mathbf{w}_0$ is given. Let f_0 be identified with f.

Algorithm 1

[1] for $k = 1, 2, \ldots$

[2] $\quad \mathbf{p}_k := -\nabla f_{k-1}(\mathbf{w}_{k-1})$;

[3] $\quad \alpha_k := \text{argmin}\{f_{k-1}(\mathbf{w}_{k-1} + \alpha \mathbf{p}_k) : \alpha \geq 0\}$;

[4] $\quad \tilde{\mathbf{w}}_k := \mathbf{w}_{k-1} + \alpha_k \mathbf{p}_k$;

[5] $\quad \mathbf{g}_k := -\nabla f_{k-1}(\tilde{\mathbf{w}}_k)$;

[6] $\quad$ Define $l_k : \mathbf{R}^n \to \mathbf{R}^n$ by $l_k(\mathbf{x}) = (I + \mathbf{p}_k \mathbf{g}_k^T / \|\mathbf{p}_k\|^2)(\mathbf{x})$;

[7] $\quad f_k := f_{k-1} \circ l_k$;

[8] $\quad \mathbf{w}_k := l_k^{-1}(\tilde{\mathbf{w}}_k)$;

[9] end

Lines [1]–[4] of this algorithm are the standard steepest descent computation. In the third line, an inexact line search may be used in place of exact minimization over α. In the sixth line, I is the $n \times n$ identity matrix. The seventh line indicates functional composition: a new objective function is formed as the composition of the old objective function and a linear change of variables.

*Department of Combinatorics and Optimization, University of Waterloo, 200 University Ave. W., Waterloo, Ontario, Canada, N2L 3G1, vavasis@math.uwaterloo.ca. Supported in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

The eighth line applies the inverse transformation to $\tilde{\mathbf{w}}_k$ so as to enforce the relationship $f_{k-1}(\tilde{\mathbf{w}}_k) = f_k(\mathbf{w}_k)$. The inverse transform is efficiently computed and applied using the Sherman-Morrison formula. Although the function value is invariant, the gradient value is not, so the algorithm is not equivalent to a sequence of steepest descent steps in the original coordinates. This algorithm is equivalent to the linear conjugate gradient algorithm in the case that f is a convex quadratic function and the line search is exact, as we shall see in Section 3.

When the algorithm terminates, say at iteration N, the vector $\mathbf{w}^{(N)}$ is a minimizer or approximate minimizer of $f^{(N)}$. Therefore, the linear transformations must be saved and applied to $\mathbf{w}^{(N)}$ in order to recover a solution to the original problem.

In certain special classes of problems, it may be feasible to implement the algorithm exactly as stated because the objective function may be accessible for updating. More commonly, however, the objective function is available only as a subroutine, in which case the algorithm must be restated in a way so that it keeps track of the linear updates itself. In particular, it must save the two vectors defining the linear transformation from all previous iterations. Then the chain rule is applied, which states that if $g(\mathbf{x}) = f(l(\mathbf{x}))$, where l is a linear function, then $\nabla g(\mathbf{x}) = l^T(\nabla f(l(\mathbf{x})))$, where l^T denotes the transposed linear function. This version of the algorithm is as follows. There is no longer a subscript on f since f is not explicitly updated in this version. The current iterate in this algorithm is denoted $\mathbf{x}_k$ and must be initialized as $\mathbf{x}_0$, which is equal to $\mathbf{w}_0$ in Algorithm 1.

Algorithm 2

[1] for $k = 1, 2, \ldots$

[2] $\quad \mathbf{p}_k := -l_{k-1}^T \circ \cdots \circ l_1^T(\nabla f(\mathbf{x}_{k-1}))$;

[3] $\quad \mathbf{m}_k := l_1 \circ \cdots \circ l_{k-1}(\mathbf{p}_k)$;

[4] $\quad \alpha_k := \operatorname{argmin}\{f(\mathbf{x}_{k-1} + \alpha \mathbf{m}_k) : \alpha \geq 0\}$;

[5] $\quad \mathbf{x}_k := \mathbf{x}_{k-1} + \alpha_k \mathbf{m}_k$;

[6] $\quad \mathbf{g}_k := -l_{k-1}^T \circ \cdots \circ l_1^T(\nabla f(\mathbf{x}_k))$;

[7] $\quad$ Define $l_k : \mathbf{R}^n \to \mathbf{R}^n$ by $l_k(\mathbf{x}) = (I + \mathbf{p}_k \mathbf{g}_k^T / \|\mathbf{p}_k\|^2)\mathbf{x}$;

[8] end

The fact that Algorithms 1 and 2 are equivalent is an easy induction. The variables $\mathbf{p}_k$, α_k and $\mathbf{g}_k$ are identical between the two algorithms, as are the sequences of linear transformations l_k. The remaining variables have the following relationships: $\mathbf{x}_k = l_1 \circ \cdots \circ l_k(\mathbf{w}_k)$ and $\mathbf{x}_k = l_1 \circ \cdots \circ l_{k-1}(\tilde{\mathbf{w}}_k)$. Note that some redundant computation in step [2] can be saved by observing that

$$\mathbf{p}_k = l_{k-1}^T(\mathbf{g}_{k-1}), \tag{1}$$

where $\mathbf{g}_{k-1}$ was computed in step [6] of the previous iteration.

We conclude this section with a result concerning the invertibility of the linear transformations.

Lemma 1. *Assume f is C^1 and none of the iterates in Algorithm 1 is a stationary point. Suppose an exact line search is used in Algorithm 1. Then l_k is invertible on every step.*

Proof. It follows from standard theory of steepest descent that if an exact line search is used, then

$$\mathbf{p}_k^T \mathbf{g}_k = 0. \tag{2}$$

(This is the first-order condition for the optimality of α for the differentiable function $f(\mathbf{w}_{k-1} + \alpha \mathbf{p}_k)$). In this case, $I + \mathbf{p}_k \mathbf{g}_k^T / (\mathbf{p}_k^T \mathbf{p}_k)$ is invertible since it follows from the Sherman-Morrison formula that $I + \mathbf{u}\mathbf{v}^T$ is invertible unless $\mathbf{u}^T \mathbf{v} = -1$. □

We remark that many kinds of inexact line searches will also yield the same result. The requirement for invertibility of l_k is that $\mathbf{p}_k^T \mathbf{p}_k \neq -\mathbf{g}_k^T \mathbf{p}_k$. Written in terms of the line search function $\phi(\alpha) = f_k(\mathbf{w}_k + \alpha \mathbf{p}_k)$, this is the same as saying that $\phi'(0) \neq -\phi'(\alpha_k)$. A line search will often enforce the condition $|\phi'(\alpha)| < |\phi'(0)|$.

Steps [2]–[3] of Algorithm 2 may be written as $\mathbf{m}_k = -H_k \nabla f(\mathbf{x}_{k-1})$, where $H_k = l_1 \cdots l_{k-1} l_{k-1}^T \cdots l_1^T$. Obviously, H_k is positive semidefinite, and assuming the condition in the previous paragraph holds, it is positive definite. This means that Algorithm 2 always produces descent directions except in the unexpected case that $\mathbf{p}_k^T \mathbf{p}_k = -\mathbf{g}_k^T \mathbf{p}_k$.

2 Specialization to quadratic functions

In this section we present some results on the specialization of Algorithm 1 to convex quadratic functions with exact line search. In particular, we prove finite termination of the algorithm. Finite termination is also a consequence of the equivalence to linear conjugate gradient (discussed in the next section), but the proof presented here is a short self-contained proof that follows different lines from customary proofs of finite termination. The difference arises from the fact that Algorithm 1 is a one-step method (i.e., it does not involve recurrences), and therefore its analysis does not require an induction hypothesis that spans the iterations as in the customary analysis.

Suppose that $f(\mathbf{w}) = f_0(\mathbf{w}) = \mathbf{w}^T A_0 \mathbf{w}/2 - \mathbf{b}_0^T \mathbf{w}$, where $A_0 \in \mathbf{R}^{n \times n}$ is symmetric and positive definite. Then it follows from step [7] of Algorithm 1 that $f_k(\mathbf{w}) = \mathbf{w}^T A_k \mathbf{w}/2 - \mathbf{b}_k^T \mathbf{w}$, where $A_k = l_k^T \cdots l_1^T A_0 l_1 \cdots l_k$ and $\mathbf{b}_k = l_k^T \cdots l_1^T \mathbf{b}_0$.

In the case of quadratic functions, the optimal choice of α_k in step [3] of Algorithm 1 is well known to be (see [1])

$$\alpha_k = \frac{\mathbf{p}_k^T \mathbf{p}_k}{\mathbf{p}_k^T A_{k-1} \mathbf{p}_k}. \tag{3}$$

We can develop the following further relationships. Combining (1) and (2) yields

$$\begin{aligned} \mathbf{p}_{k+1} &= l_k^T(\mathbf{g}_k) \\ &= \left(I + \frac{\mathbf{g}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{p}_k} \right) \mathbf{g}_k \\ &= \mathbf{g}_k. \end{aligned}$$

Also,

$$\begin{aligned} \mathbf{g}_k = \mathbf{p}_{k+1} &= -\nabla f_{k-1}(\tilde{\mathbf{w}}_k) \\ &= -A_{k-1} \tilde{\mathbf{w}}_k + \mathbf{b}_{k-1} \end{aligned}$$

$$
\begin{aligned}
&= -A_{k-1}\mathbf{w}_k + \mathbf{b}_{k-1} - \alpha_k A_{k-1}\mathbf{p}_k \\
&= \mathbf{p}_k - \alpha_k A_{k-1}\mathbf{p}_k.
\end{aligned}
$$

With these relationships in hand, we can now propose the main result that implies to finite termination.

Let us introduce the following notation:

$$
K(A_{k-1}, \mathbf{p}_k) = \text{span}(\mathbf{p}_k, A_{k-1}\mathbf{p}_k, A_{k-1}^2\mathbf{p}_k, \ldots),
$$

i.e., the minimal invariant subspace of A_{k-1} that contains $\mathbf{p}_k$.

Theorem 1. *The invariant subspace $K(A_k, \mathbf{p}_{k+1})$ is a proper subspace of $K(A_{k-1}, \mathbf{p}_k)$.*

Proof. First, observe that $\mathbf{p}_{k+1}(= \mathbf{g}_k) \in K(A_{k-1}, \mathbf{p}_k)$, which follows from equality demonstrated above that $\mathbf{p}_{k+1} = \mathbf{p}_k - A_{k-1}\mathbf{p}_k$. Next, we claim more generally that $K(A_k, \mathbf{p}_{k+1}) \subset K(A_{k-1}, \mathbf{p}_k)$. This follows because $A_k = l_k^T A_{k-1} l_k$. The three operators l_k, l_k^T and A_{k-1} all map $K(A_{k-1}, \mathbf{p}_k)$ into itself since $\mathbf{g}_k$ and $\mathbf{p}_k$ are both already proven to lie in this space. Thus, A_k maps $K(A_{k-1}, \mathbf{p}_k)$ into itself.

Thus, we have shown $K(A_k, \mathbf{p}_{k+1}) \subset K(A_{k-1}, \mathbf{p}_k)$. To conclude the proof, we must show that it is a proper subspace. We claim that $K(A_k, \mathbf{p}_{k+1}) \subset \mathbf{p}_k^\perp$. Observe first that $\mathbf{p}_{k+1} \in \mathbf{p}_k^\perp$; this follows immediately from (2). Next, it is obvious from the definition of l_k that $\mathbf{p}_k$ is a right eigenvector of l_k. Furthermore, $\mathbf{p}_k$ is a right eigenvector of $l_k^T A_{k-1}$, as we see from the following algebra:

$$
\begin{aligned}
l_k^T A_{k-1}\mathbf{p}_k &= \left(I + \frac{\mathbf{g}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{p}_k}\right) A_{k-1}\mathbf{p}_k \\
&= A_{k-1}\mathbf{p}_k + \mathbf{g}_k \frac{\mathbf{p}_k^T A_{k-1}\mathbf{p}_k}{\mathbf{p}_k^T \mathbf{p}_k} \\
&= A_{k-1}\mathbf{p}_k + \mathbf{g}_k/\alpha_k \\
&= A_{k-1}\mathbf{p}_k + (\mathbf{p}_k - \alpha_k A_{k-1}\mathbf{p}_k)/\alpha_k \\
&= \mathbf{p}_k/\alpha_k.
\end{aligned}
$$

The statement under consideration $K(A_k, \mathbf{p}_{k+1}) \subset \mathbf{p}_k^\perp$ can be rewritten as the equation $\mathbf{p}_k^T A_k^i \mathbf{p}_{k+1} = 0$ for all i, i.e., $\mathbf{p}_k^T (l_k^T A_{k-1} l_k)^i \mathbf{g}_k = 0$. But $(l_k^T A_{k-1} l_k)^i$ can be factored as products of l_k^T and $A_{k-1} l_k$, and we have just proved that $\mathbf{p}_k^T$ is a left eigenvector of both of these operators. Therefore, $\mathbf{p}_k^T (l_k^T A_{k-1} l_k)^i \mathbf{g}_k = \text{scalar} \cdot \mathbf{p}_k^T \mathbf{g}_k = 0$.

Therefore, we have proved that

$$
K(A_k, \mathbf{p}_{k+1}) \subset K(A_{k-1}, \mathbf{p}_k) \cap \mathbf{p}_k^\perp.
$$

Thus, to show that $K(A_k, \mathbf{p}_{k+1})$ is a proper subset of $K(A_{k-1}, \mathbf{p}_k)$, it suffices to show that $K(A_{k-1}, \mathbf{p}_k)$ is not a subspace of $\mathbf{p}_k^\perp$. But this is obvious, since the former contains $\mathbf{p}_k$ while the latter does not. □

This theorem proves finite termination of Algorithm 1: the dimension of the invariant subspace at iteration 0 is at most n, and the dimension shrinks by at least 1 each iteration, so therefore the algorithm terminates in at most n iterations.

More strongly, if the coefficient matrix A_0 has at most s distinct eigenvalues, then Algorithm 1 terminates in at most s iterations, since any vector lies in an invariant subspace of dimension at most s for such a matrix.

Finally, the above theorem suggests that Algorithm 1 converges superlinearly. We recall the following two facts (see [1]): the steepest descent algorithm applied to a convex quadratic function converges at a rate proportional to the condition number of the matrix. Furthermore, the condition number of a matrix acting on a subspace can never exceed (and is usually less than) the condition number of the matrix acting on the whole space, a consequence of the Courant-Fisher minimax theorem. Thus, we see that Algorithm 1 consists of steepest descent in ever smaller invariant subspaces, so the effective condition number of the matrix decreases (or at least, does not increase) each iteration and hence the convergence rate is expected to be superlinear.

3 Equivalence to linear conjugate gradient

Again, we assume for this section that $f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}/2 - \mathbf{b}^T\mathbf{x}$, where $A \in \mathbf{R}^{n\times n}$ is symmetric and positive definite. We assume again that the line search is exact. We prove that Algorithm 2 is equivalent to linear conjugate gradient. For the sake of completeness, let us write linear conjugate gradient in its usual form as follows. Let $\mathbf{x}_0$ be given.

Algorithm Linear-CG

[1] $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$;
[2] for $k = 1, 2, \ldots$
[3] if $k = 1$
[4] $\mathbf{n}_1 = \mathbf{r}_0$;
[5] else
[6] $\beta_k = \mathbf{r}_{k-1}^T\mathbf{r}_{k-1}/(\mathbf{r}_{k-2}^T\mathbf{r}_{k-2})$;
[7] $\mathbf{n}_k = \beta_k\mathbf{n}_{k-1} + \mathbf{r}_{k-1}$;
[8] end
[9] $\alpha_k = \mathbf{r}_{k-1}^T\mathbf{r}_{k-1}/(\mathbf{n}_k^T A\mathbf{n}_k)$;
[10] $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{n}_k$;
[11] $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k A\mathbf{n}_k$;
[12] end

Well known properties of Linear-CG are that $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = -\nabla f(\mathbf{x}_k)$ and that the $\mathbf{r}_k$'s are mutually orthogonal (see [1]). We claim that Algorithm 2 and Algorithm Linear-CG are equivalent with the following relationships among the variables: $\mathbf{p}_k = \mathbf{r}_{k-1}$; $\mathbf{m}_k = \mathbf{n}_k$; $\mathbf{g}_k = \mathbf{r}_k$, and α_k is the same between the algorithms. This equivalence is proved by induction. For the $k = 1$ case, it is clear that $\mathbf{p}_1 = \mathbf{m}_1 = \mathbf{n}_1 = \mathbf{r}_0$ and $\mathbf{g}_1 = \mathbf{r}_1$. For $k > 1$, we see that

$$\begin{aligned}\mathbf{p}_k &= -l_{k-1}^T \cdots l_1^T \nabla f(\mathbf{x}_{k-1}) \\ &= \left(I + \frac{\mathbf{g}_{k-1}\mathbf{p}_{k-1}^T}{\mathbf{p}_{k-1}^T\mathbf{p}_{k-1}}\right) \cdots \left(I + \frac{\mathbf{g}_1\mathbf{p}_1^T}{\mathbf{p}_1^T\mathbf{p}_1}\right)\mathbf{r}_{k-1}\end{aligned}$$

$$
\begin{aligned}
&= \left(I + \frac{\mathbf{r}_{k-1}\mathbf{r}_{k-2}^T}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}}\right)\cdots\left(I + \frac{\mathbf{r}_1\mathbf{r}_0^T}{\mathbf{r}_0^T\mathbf{r}_0}\right)\mathbf{r}_{k-1} \\
&= \mathbf{r}_{k-1}.
\end{aligned}
$$

The second and third line both involved application of the induction hyptohesis, and the last line follows because all terms drop out from the product with $\mathbf{r}_{k-1}$ except the identity because the $\mathbf{r}_i$'s are mutually orthogonal.

Next, we show by induction that $\mathbf{n}_k = \mathbf{m}_k$. Observe from step [7] of Linear-CG that $\mathbf{n}_k - \beta_k\mathbf{n}_{k-1} = \mathbf{r}_{k-1}$ while

$$
\begin{aligned}
\mathbf{m}_k - \beta_k\mathbf{m}_{k-1} &= \mathbf{m}_k - \left(\frac{\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}}\right)\mathbf{m}_{k-1} \\
&= l_1\cdots l_{k-1}\mathbf{p}_k - \left(\frac{\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}}\right) l_1\cdots l_{k-2}\mathbf{p}_{k-1} \\
&= l_1\cdots l_{k-2}\left(l_{k-1}\mathbf{r}_{k-1} - \left(\frac{\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}}\right)\mathbf{r}_{k-2}\right) \\
&= l_1\cdots l_{k-2}\left(\mathbf{r}_{k-1} + \frac{\mathbf{r}_{k-2}\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}} - \left(\frac{\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T\mathbf{r}_{k-2}}\right)\mathbf{r}_{k-2}\right) \\
&= l_1\cdots l_{k-2}(\mathbf{r}_{k-1}) \\
&= \mathbf{r}_{k-1}.
\end{aligned}
$$

In the above derivation, we applied the induction hypothesis, the definition of l_{k-1}, and, for the last line, again the fact that the $\mathbf{r}_i$'s are mutually orthogonal. This equation proves that $\mathbf{n}_k - \beta_k\mathbf{n}_{k-1} = \mathbf{m}_k - \beta_k\mathbf{m}_{k-1}$, hence the sequence of $\mathbf{m}_k$'s and $\mathbf{n}_k$'s are equal. Finally, we must claim that $\mathbf{g}_k = \mathbf{r}_k$. Again, this follows from step [6] of Algorithm 2 and the orthogonality of the $\mathbf{r}_k$'s.

4 The secant condition

In this section we drop the assumption that f is quadratic but continue to assume that it is C^1. We prove that if the line search is exact, then Algorithm 2 satisfies the secant condition, which states

$$H_{k+1}\mathbf{y}_k = \mathbf{m}_k$$

where $H_{k+1} = l_1 \circ \cdots \circ l_k \circ l_k^T \circ \cdots \circ l_1^T$, that is, the operator that carries $-\nabla f(\mathbf{x}_k)$ to $\mathbf{m}_{k+1}$, and $\mathbf{y}_{k+1} = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. The secant condition is usually stated as the requirement that $H_{k+1}\mathbf{y}_k = \alpha_k\mathbf{m}_k$ [4]. The scaling factor, however, is inconsequential because the algorithm can be equivalently presented with a different scaling of H_k; that scaling would be canceled in the line search, which would carry out the reciprocal scaling.

It should be noted that the best known secant algorithms including DFP and BFGS satisfy the secant condition regardless of whether the line search is exact, so Algorithm 2 differs from these algorithms in this respect.

Checking the secant condition is fairly straightforward algebra as follows. It follows from steps [2] and [6] that $l_{k-1}^T\circ\cdots\circ l_1^T(\nabla f(\mathbf{x}_{k+1})) = -\mathbf{g}_k$ and $l_{k-1}^T\circ\cdots\circ l_1^T(\nabla f(\mathbf{x}_k)) = -\mathbf{p}_k$,

hence

$$l_{k-1}^T \circ \cdots \circ l_1^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) = \mathbf{p}_k - \mathbf{g}_k.$$

Next, applying l_k^T yields:

$$\begin{aligned} l_k^T l_{k-1}^T \circ \cdots \circ l_1^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) &= l_k^T (\mathbf{p}_k - \mathbf{g}_k) \\ &= \left(I + \frac{\mathbf{g}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{p}_k} \right) (\mathbf{p}_k - \mathbf{g}_k) \\ &= \mathbf{p}_k - \mathbf{g}_k + \frac{\mathbf{g}_k \mathbf{p}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{p}_k} - \frac{\mathbf{g}_k \mathbf{p}_k^T \mathbf{g}_k}{\mathbf{p}_k^T \mathbf{p}_k} \\ &= \mathbf{p}_k, \end{aligned}$$

where, to obtain the last line, we invoked (2) since the line search is exact. Next, since $\mathbf{p}_k$ is an eigenvector of l_k with eigenvalue 1 (again using the fact that the line search is exact so $\mathbf{g}_k^T \mathbf{p}_k = 0$),

$$l_k l_k^T l_{k-1}^T \circ \cdots \circ l_1^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) = \mathbf{p}_k.$$

Finally, applying $l_1 \cdots l_{k-1}$ to both sides and applying statement [3] yields the desired result.

5 Computational results (preliminary)

In this section we compare Algorithm 2 to BFGS, DFP, Polak-Ribière conjugate gradient (CG-PR+), and Fletcher-Reeves conjugate gradient (CG-FR). Refer to [4] for information about all of these algorithms. In this section we denote Algorithm 2 as SDICOV for "steepest descent with iterated change of variables." We report only the number of iterations. The BFGS and DFP algorithms are implemented using product form rather than explicit formation of H_k. This means that, like SDICOV, the number of operations and storage requirement for the kth iteration is $O(kn)$ plus a function and gradient evaluation (plus additional function and gradient evaluations in the line search). In contrast, CG-PR+ and CG-FR require only $O(n)$ storage and $O(n)$ operations per iteration. Therefore, the iteration counts reported here partially hide the greater efficiency of CG-PR+ and CG-FR.

The first test is the nonconvex *distance geometry* problem [2], a nonlinear least squares problem. There are n particles in $\mathbf{R}^2$ whose positions are unknown. One is given the interparticle distances for some subset of possible pairs of particles. The problem is to recover the coordinates from these distances. Thus, the unknowns are $\mathbf{x}_3, \ldots, \mathbf{x}_n$, positions of particles 3 to n, each a vector in $\mathbf{R}^2$. To remove degenerate degrees of freedom, we assume the positions of particles 1 and 2 are fixed. The objective function is

$$f(\mathbf{x}_3, \ldots, \mathbf{x}_n) = \sum_{(i,j) \in E} (||\mathbf{x}_i - \mathbf{x}_j||^2 - d_{ij}^2)^2$$

where E denotes the subset of $\{1, \ldots, n\}^2$ of pairs whose distance is given and d_{ij} denotes the given distance. This problem has multiple local minima (indeed, global minimization

Table 1: Results of distance geometry trials

Algorithm	Ave. no. iterations	
	$n_{\text{particle}} = 10$	$n_{\text{particle}} = 100$
SDICOV	34	76
BFGS	20	75
DFP	24	80
CG-PR+	93	107
CG-FR	146	161

of this function is known to be NP-hard), so the testing procedure must account for the possibility that that different algorithms could converge to different minimizers, which could skew iteration counts. To avoid this possibility, we constructed instances with a known global minimizer (by first selecting the positions randomly, and then computing the interpair distances from those positions). Then we initialized the algorithm fairly close to the global minimizer so that all algorithms would fall into the same basin. The line search procedure is inexact: it uses bisection with a termination criterion that $|\phi'(\alpha)| \leq 0.2|\phi'(0)|$. The convergence tolerance is a relative reduction in the norm of the gradient of 10^{-5}. Two sizes were tried, namely 10 particles ($n = 16$) and 100 particles ($n = 196$). For each problem size, four trials were run, and the number of iterations over the trials was averaged. The results are summarized in Table 1. For the smaller problem SDICOV was worse than BFGS or DFP, but for the larger problem, the three algorithms have similar performance. The two versions of conjugate gradient are slower.

The second test is a larger class of problems, namely, a finite element mesh improvement problem. Given a subdivision of a region $\Omega \subset \mathbf{R}^3$ into tetrahedra, the problem under consideration is to displace the nodes of the tetrahedra in such a way as to improve the overall quality of the mesh. There are several measures of quality; we use the ratio of the volume of the tetrahedra to the cube of one of its side lengths. The minimum such ratio over all tetrahedra is a measure of the mesh quality (the closer to 0, the worse the mesh). The details of our method are in [5]. Briefly, we smooth this nonsmooth unconstrained problem (nonsmooth because it is maximization of a minimum) by introducing an auxiliary variable standing for the minimum ratio and constraints to enforce its minimality. The smoothed constrained problem is the solved with a barrier function approach. Ultimately, the problem reduces again to an unconstrained problem, except the objective function is a smoothed version of the original that involves the logarithms of the ratios.

There is a second source of nondifferentiability that remains in the problem due to parametrization of the boundary. For interior nodes in the mesh, the variables in the optimization problem are its (x, y, z) coordinates. For nodes on the boundary, however, the variables are the (u, v) or t parametric coordinates of the boundary surface. We wish to allow nodes on the boundary to move from one parametric patch of a boundary surface to another; such movement introduces a nondifferentiable jump in the objective function. If the boundary surfaces are smooth, it would be possible in principle to come up with smooth local parametrizations that would circumvent this difficulty, but we have not done

Table 2: Results of the optimization algorithms on the mesh improvement problem. A missing entry indicates failure of the iteration.

	No. of iterations		
	Cylinder	Large cavity	Small cavity
n	7380	8775	4254
SDICOV	22	87	373
BFGS	—	—	—
DFP	—	—	—
CG-PR+	34	—	—
CG-FR	81	—	—

so.

The line search is again based on bisection and enforces the inequality $|\phi'(\alpha)| \leq 0.7|\phi'(0)|$. It needs a safeguard, since a step too large can invert a tetrahedron, thus sending the above ratio to a negative number and hence making the logarithm undefined. The initial point for the optimization routine is the mesh produced by the QMG mesh generator [3].

We tested three problems: a mesh of a cylinder, of a cube with a large spherical cavity, and of a tetrahedron with a small octahedral cavity. For this third problem, each boundary surface is a single flat parametric patch, so the problem is differentiable because there are no parametric jumps. The results of this test are shown in Table 2. This problem is again nonconvex and probably has many local minima. In this test case, we did not have a means to ensure that the different algorithms find the same minimizer. The algorithms, however, returned solutions with comparable objective function values (when they succeeded). BFGS and DFP failed in every case in the sense that they terminated due to stagnation prior to satisfaction of the convergence termination criterion. Our test for stagnation was four successive iterations without significant reduction in either the function value or gradient norm. Prior to stagnation, there was generally slow progress in these algorithms; for example, the stagnation test required 127 iterations for BFGS and 126 for DFP in the cylinder case to activate. The two conjugate gradients also sometimes failed due to stagnation; CG-PR+ also failed once for producing a search direction that was not a descent direction.

It must be pointed out that we have not implemented a restart strategy for either BFGS or DFP. Most modern implementations would have such a strategy, and this would presumably ameliorate the difficulty with slow progress.

6 Concluding remarks

We propose a new iterative method for unconstrained minimization. The algorithm is based on steepest descent after a linear change of coordinates. It is a secant method if the line search is exact. It always produces a descent direction except in the case that the line search produces a certain degenerate result.

In practice, the new method works well on two test cases. Our preliminary results hint that, if restarting is not used, the new algorithm in practice is sometimes more robust than BFGS and DFP.

References

[1] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* John Hopkins University Press, 3rd edition, 1996.

[2] B. Hendrickson. The molecule problem: exploiting structure in global optimization. *SIAM J. Optimization*, 5:835–857, 1995.

[3] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Computing*, 29:1334–1370, 2000.

[4] J. Nocedal and S. Wright. *Numerical Optimization, 2nd Edition.* Springer, New York, 2006.

[5] Shankar Prasad Sastry, Suzanne Shontz, and Stephen Vavasis. A new mesh quality improvement method based on a log-barrier formulation. In preparation, 2008.